Front End Development and System Test Case Generation for Collaborative Invention Mining

Jyoti Prakash Meher

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

Front End Development and System Test Case Generation for Collaborative Invention Mining

Dissertation submitted in

May 2014

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Jyoti Prakash Meher

(Roll 212CS3124)

under the supervision of

Dr. Durga Prasad Mohapatra

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

dedicated to Lord Jagannath and Beloved friends...

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela-769 008, Odisha, India.

May 2014

Certificate

This is to certify that the work in the thesis entitled ***"Front End Development ans System Test Case Generation for Collaborative Invention Mining"*** by ***Jyoti Prakash Meher***, bearing roll number **212CS3124**, is a record of an original research work carried out by him under my supervision and guidance in partial fulfilment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Dr. Durga Prasad Mohapatra
Associate Professor
Dept. of CSE
NIT, Rourkela

Acknowledgement

This dissertation, though an individual work, has benefited in various ways from several people. Whilst it would be simple to name them all, it would not be easy to thank them enough.

The enthusiastic guidance and support of *Prof. Durga Prasad Mohapatra* inspired me to stretch beyond my limits. His profound insight has guided my thinking to improve the final product. My solemnest gratefulness to him.

I am also grateful to *Prof. Santanu Kumar Rath* Head, Department of Computer Science and Engineering for his ceaseless support throughout my research work. My sincere thanks to *Prof. Bansidhar Majhi*, *Dr. Debi Prasad Swain*, and *Dr. Suman Bhattacharya* for their continuous encouragement and invaluable advice.

It is indeed a privilege to be associated with people like *Prof. S. K. Jena*, *Prof. P. M. Khilar*, *Prof. A. K. Turuk*, *Prof. S.Chinara*, *Prof. B. D. Sahoo* and *Prof. Pankaj Sa*. They have made available their support in a number of ways.

Many thanks to my comrades and fellow research colleagues. It gives me a sense of happiness to be with you all. Special thanks to my beloved friends whose support gave a new breath to my research.

Finally, my heartfelt thanks for her unconditional love and support. Words fail me to express my gratitude to my beloved parents who sacrificed their comfort for my betterment.

Jyoti Prakash Meher

Abstract

System testing plays a vital role to ensure software quality assurance and software quality control. It is possible to minimize the development time by parallely executing the software development process as well as testing process. In a typical Software development methodology, almost 60% of development effort is spent in testing phase itself so as to increase the reliability of the product.

The UML is a design model which can describe the dynamic behavior of a system. So, it can be considered as a tool for testing as it behaves as a simulation model. The activity diagram represents the system as a whole. Hence, it has become convenient to consider activity diagram for system testing.

We have designed the front end components of the application by using Adobe Flex 3.0 technology. To design this, we have followed the business requirement documents.

Here, we have considered the UML activity diagram of Collaborative Invention Mining (CIM) to generate the system test cases from it. Initially we have taken the activity diagram as input and applied an algorithm called Activity Path Traversal (APT) to generate the test paths from it. The finally we take the generated test paths as input and applied an algorithm called Test Path Traversal (TPT) to generate the system test cases from it. We have also used an tool called GraViz Editor to validate the intermediate paths generated from the first algorithm.

Finally, we have compared the generated system test cases with the system test cases designed by the test team of the Industry leading to an optimized set of system test cases.

***Keywords*:** CIM, Activity diagram, system test cases, UML, etc.

Contents

List of Figures

List of Tables

Chapter 1

Inroduction

Testing a software is an arrangement of an atomic unit of work with a purpose of assessing the characteristics or functionalities of a system or framework and guaranteeing that it reaches its envisaged results. As per ANSI/IEEE 1059 standard, Testing might be characterized as "A procedure of breaking down a system to locate the contrasts between existing and obliged conditions (that is a deviation/gaps/bugs) and to assess the characteristics of the software component". Albeit essential for programming quality assurance, testing a software still remains an art, because of restricted understanding of the business requirement of the software. The trouble in testing a system stems from the entanglement of system: we can't totally test a system with moderate complicacy. The expectation of testing could be quality certification, assurance and acceptance, or estimation of reliability etc. System testing is a business among estimated budget, time and quality. Testing can never distinguish all the imperfections inside application totally. Rather, it outfits a feedback that looks at the state and conduct of the item against a prophet, i.e., standards or approaches by which somebody may comprehend the issue. These prophets may incorporate programming particulars, business necessity, practical identical items, past forms of the same item (if any), derivations about expected reason, customer or client desires, applicable advancement norms, etc.

1.1 Motivation of our Work

So far as the era is recognized as the universe of rising pattern, the customer or client needs to persuade products to be actualized and updated quicker than others. As the following version of the product will be delivered in next few days, and gets just several days of testing before it is delivered. So because of this brief time period or consistent delivery, the more bugs get heaped up into the product and which gets altered in the following release. A software released with a variety of bugs into it might influence the users and the clients experience which makes a terrible effect on quality impression of your organization's brand. To guarantee the product quality assurance and quality control, testing a system is recognized as an essential period of the Software Development Life Cycle (SDLC).

In the Software Development Life Cycle (SDLC), the Testing process is assumed to be a key part, which serves to enhance the quality, performance and reliability of the system that fulfills the business and practical necessities of the customers and/or clients. In this way, it is better to present testing in the early phase of the SDLC stages so it serves to recognize the bugs in the early stage and attempt to safeguard the bugs discovering and get it determined get it resolved as early as possible so as to optimize the scheduled time and development cost. In the testing methodology, it may not fix all defects reside in the application or we can't say that the application is 100% defect free, yet taking one stage ahead to doing this and give ease of use.

Testing a software product is an investigative methodology led to give information about the nature of the product or system under test to the system stakeholders. System testing can additionally give environment, free perspective of the requisition that permits the business to acknowledge and comprehend the risks of the product development.

1.2 Objectives of our Work

The main objective of our research work is to implement the front end of a focused program coupled with management commitment to encourage Inventions (and Innovations) from different geographical locations is essential for survival of any business and to generate system test case automatically from the design document i.e., Activity diagram so as to optimize the cost and time of the testing effort. To address this objective, we identify the following goals:

- To construct the UML diagrams of the application.
- To develop the front end components of the software.
- To generate test scenarios from its design document, i.e., Activity Diagram and generate test data for the path.
- To generate the basic test paths from Activity diagram.
- To validate the test paths by generating an intermediate representation called Control Flow Graph (CFG) using GVEdit 2.26.
- To design the system test cases from the test paths generated automatically.
- And finally, to compare the generated system test cases with the existing test case designed by the testing team of the Organization.

1.3 Organization of Thesis

Our thesis is divided into six chapters, including the current chapter and each chapter is orchestrated as below:

Chapter 2 represents some basic concepts and ideas regarding the system testing and related work. It contains testing types, testing approaches, testing levels, etc.

Chapter 3 depicts a survey of the related work regarding the test case generation from from the UML Activity diagram. Here we have accumulated some basic notions on test case generation.

Chapter 4 provides a brief idea regarding the development of Software system termed as Collaborative Invention Mining that has been implemented by the Company. Here we discuss the application development methodology of the application to the test case designed for the same.

Chapter 5 describes the test case generation approach from the design document, i.e., Activity diagram of the application. It includes generation of test paths with intermediate graphical representation of the paths from which test cases have been generated. We also introduced some basic concepts about the UML diagrams giving more emphasis on activity diagrams.

Chapter 6 represents the result generated from the implemented methodology. We compare the generated test cases with the test cases designed by the testing team of the Industry. Finally, we conclude the research with a summary of our contribution harbingering a possible future extension of our work in this direction.

Chapter 2

Basic Concepts

Here, we will discuss some basic concepts about software testing that we have already studied in our previous course of study. Apart from that we will discuss the motivations and objectives of the research work.

2.1 Software Testing

Testing a system is basically referred as manipulating it with an intend to find the deviation from its expected result. Its main goal is to detect the defects present, if any, in the system so as to increase its reliability by fixing it. But, testing cannot assure tat an application or system is fully defect free, i.e., it will work properly on every environment. It may provide a certain environment where we can detect defects in the system. The scope of testing depends on its types where, In white box testing, it investigates the code while in black box testing, it executes the system. Nowadays, the testing team is bifurcated from the development team in almost every software enterprise. The types of defects detected in a system not only make the product reliable, but also makes the development team efficient and more productive.

In the current software development trend, a testing team may be differentiated from the development team. There are different parts for testing allies. Information determined from system testing may be utilized to redress the system

by which it is created.

Software testing can be stated as the process of validating and verifying that a software program/application/product:

- meets the requirements based on which the design and development is being carried out,
- results as expected,
- and satisfies the needs of the stakeholders of the application.

In this way, testing a system might be developed at whenever in the development process of software relying upon the testing process being utilized in it. Customarily, a large portion of the test exertion happens after the business requirement have been solidified and the coding process has been finished, yet in the Agile methodology, the greater part of the test exertion is on-going parallel.

2.2 Testing Approaches

2.2.1 Static Testing

In Static testing the code is manually verified by the experts instead of executing the code where the documents may be Software requirement specification, and design documents to find errors in itself.

The prime target of static testing is to enhance the nature of software quality by discovering lapses in right on time phases of the development life cycle. This testing is likewise called as Non-execution strategy or verification. Static testing includes manual or mechanized survey of the records in regards to the development of applications. It checks reports and gives review comments on it.

There are many approaches to testing among which review, walk-through or inspection are considered as static testing which are illustrated as below.

- Informal Reviews: This is a process of evaluating the document which does not follow any process to find out any error. Here the documents are only reviewed and some informal review comments are given on it.

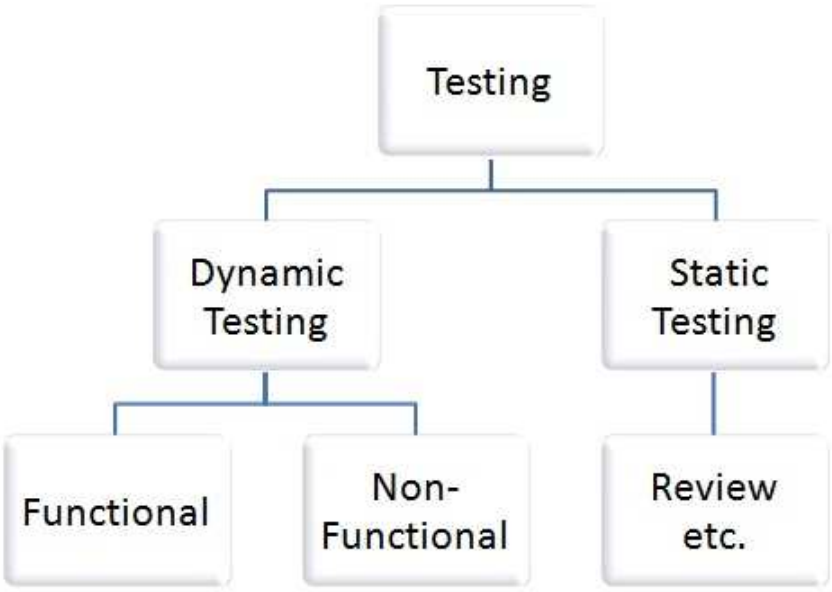


Figure 2.1: Testing Approaches

- Technical Reviews: Here the technical specification of the application software is reviewed by a peer team to ensure that whether the it is suitable for the project or not.They try to find any deviance in the specifications and standards followed. This review concentrates mainly on the technical documents related to the software such as Test Strategy, Test Plan and requirement specification documents.

- Walk-through: Here the product is explained by the product manager of the product for his team. A questioned based discussion is being carried out where different questioned are being collected from different team members. Scribe makes the review comments.

- Inspection: The main objective of the inspection is to find defects, if any, in the documents which is led by a trained moderator. It is a formal review where the document is strictly reviewed by following a strict process so as to maximize the number of defects. The review process is carried out by the help of a review check list which records the defects found and informs the

participants to verify and fix the errors.

- Static code Review: This is a process of systematic review of the source code of the software without executing the codes. It includes the coding standards, the syntax of the code written, code optimization et al. Sometimes it also considered as white box testing.

In this way the static testing can be done at any stage of the software development.

2.2.2 Dynamic Testing

Dynamic testing is known as the process of executing the code of an application with a given set of test cases that has been designed. Dynamic testing occured at the time of program in execution. Basically, unit testing and integration testing are done in this category. It may begin with an incomplete block of code. This can be applied to the simultaneous discrete module. In integration testing, it uses stubs and drivers to make the block of code to complete it.

2.3 The Box Approach

Testing a software can be achieved in many ways, one of which is the box approach in which, it is divided into two parts called as the white box and the black box approach. This categorization is basically done on the basis of the anatomy of the application, i.e., it may be with complete internal structure or may be only the external structure.

2.3.1 White-box testing

The process of testing the anatomy of the application or working of the application with its internal details. To achieve this process, the tester should have prime knowledge about the internal structure of the application. In other words, we may say that the tester should be a developer first. This covers the unit and

integration testing process..
Techniques used in white-box testing include:

- Code coverage code coverage is satisfied by creating some code criteria, which includes statement coverage, branch coverage, path coverage et al.
- Mutation testing methods, etc.

Hence, all the paths or branches are executed at least once and this can be achieved by 100% statement coverage only which is helpful in fixing the functionality errors in the application. But, its not a sufficient condition as the input environment is not fixed.

2.3.2 Black-box testing

Evaluating the functionalities without any knowledge of the internal implementation of the application is termed as black-box testing. It treats the software as a black-box. The tests focus on what the software is supposed to do rather how it does. This method includes equivalent partitioning, boundary value analysis, state transition tables, decision tables, model-based testing, ad-hoc testing et al.One advantage of the black box technique is that programming knowledge is not required.

Black box testing applies to all levels of software testing such as unit testing, integration testing, system testing and acceptance testing.

Exploratory testing and Ad hoc testing are important testing methodologies to check software integrity, because a less time for preparation is required to implement this, where the important bugs can be found quickly. In ad hoc testing, where testing takes place in an improvised, impromptu way.

2.4 Testing Levels

Test levels are categorized into four types according to their level of complexity of the code. Among these one is integration testing. the second one is system

testing and the third one is user acceptance testing. Tests are frequently grouped by where they are integrated in the software development process, or by the level of specificity of the test.

2.4.1 Unit Testing

These sorts of tests are typically composed of software engineers as they are complying with codes, to guarantee that the particular capacity is functioning of course. One capacity may contain different tests, to get a different branch in the code. Unit testing alone can't check the usefulness of a block of code, rather, is utilized to guarantee that the building hinders the software users a free function of one another.

Unit testing is a part of software implementation that includes synchronized provision of a wide range of deformity discovery and anticipation procedures so as to diminish implementation risk, time, and software development expenses. It is performed by the developer in a software enterprise throughout the implementation phase of the software development Life cycle. Instead of supplant customary QA focuses, it encourages it. Unit testing expects to wipe out development failures before code is elevated to the QA team. This method should increment the nature of the product come about and in addition the productivity of the general development procedure and quality assurance.

Contingent upon the desire of organization for development of software, unit testing may incorporate static analysis of code, data flow analysis, matrix analysis, peer code review, code coverage analysis and other software assurance hones.

2.4.2 Integration Testing

Integration testing is the process of testing the integrated modules. It is used in the v model approach of testing, in which it comes after the unit testing. Basically, this type of testing is carried out by the testers so as to check the interfaces among the different modules. In this type of testing, the defects related to module interfacing is detected. It can be considered as a part of both development and testing phase.

2.4.3 System Testing

After a system is completely integrated and handed over to the testing team for testing, the system testing begins by the testing team. It is basically considered as black box testing as the testing team does not have prior knowledge about the development structure of the application. In this testing, the business requirements are met. Sometimes it is also called end-to-end testing as it covers the whole system

2.5 Software Quality Assurance and Quality Control (SQA/SQC)

Testing software strengthens software quality assurance and quality control (SQA/SQC) handle in a software industry. In SQA/SQC, software analysts and business analysts concern for the software development procedure instead of simply the depicting for example, documentation, code et al. They research and adjust the software implementation methodology itself to decrease the numbers of gaps that comes about in the conveyed product which is called defect rate.

What constitutes an "acceptable defect rate" relies on upon the characteristics of the product, for instance, a flight simulator program can have much higher defect tolerance limit than programming for an actual airplane. In spite of the fact that there are close connections with SQA/SQC, testing team regularly exists autonomously, and there may be no SQA/SQC work in a few organizations depending on the sorts of software projects they are executing.

Testing the software is an errand expected to detect faults in software by challenging a computer program expected results with its actual results for a given set of inputs. By complexity, SQA/SQC is the approaches for usage and strategies with a proposition to keep defects from happening in the first endeavor.

Chapter 3

Literature Survey

In this chapter we have reviewed a variety of survey papers, publications and journals regarding our thesis work. This chapter is categorized into two parts where first part represents the survey about Collaborative invention mining and the second section represents the survey about generating test cases from different uml diagrams of the design documents.

3.1 Collaborative Invention Mining

Invention Mining is the process of working incessantly so as to discover potential inventions as early as possible during the development Lifecycle. This process makes the invention matured So achieving this process by a group of people belonging to different geographical location is termed as Collaborative Invention Mining. So as to automate this process, much more research is being carried out by different people in the world to propose such an application over the internet.

Jim Anderson [1] has proposed an approach for internet data mining collaboratively from different geographical locations that facilitate a group to automatically process the information provided by the guides and thereby creates a brand and a charming look and feel to the web site supported by the plurality of the groups. Subsequently, a system for mining and strengthening an invention or idea is implemented by Santosh Mohanty which is termed as Collaborative

Invention Mining (CIM) [2]. Here a new ideation data are being processed through a 48-cell Idea Detailing Tree (IDT) that systematically validates and matures it in an iterative manner. The cells are filled in different stages of the application leading to a final score. Here, a score of 450 is considered as a threshold value, i.e., if an idea carries a score more that 450 then it can be patentable otherwise it may not.

3.2 Test case generation from UML diagrams

In [2], a generous test cases are generated from a system under test by using a Java program. After that the program is run with the generated test cases to get a corresponding test path. Finally, these generated test paths are compared with the activity diagram which implies the coverage criteria [3]. This approach can also be used to draw a general idea about the behavior of the activity diagram and the program execution.

Kundu and Samant [4] have considered the activity diagram to generate test cases. Initially, they have drawn the activity diagram with necessary test data the converted the diagram into a graph called an activity graph. And finally, generated the test cases from the activity graph generated above .

Mishra [5] have proposed a methodology for embedded system. Here, they have considered the activity diagram for test case generation. They have used the specification coverage criteria which used to generate system test cases for embedded systems. This approach is helpful in reducing the efforts for validation not only in specification level but also in implementation level.

Xu *et al.* [6] has proposed an automatic approach to generate test case by using activity diagram. They have used adaptive agents to achieve this methodology. [7]

Kim *et al.* [9] has considered th activity diagram to generate test cases as it represents the dynamic behavior of a system by interaction of different objects among them selves. Here, the number of test cases are minimized depending upon their applicability in the system. Initially they have drawn an input out explicit activity diagram from which, they generated a directed graph. Form this

graph, the test cases are generated.This methodology avoids the problem of stste explosion by using principle of single stimulus to represent the dynamic behavior of the system.

Chapter 4

Collaborative Invention Mining (CIM)

4.1 Introduction

A framework of mining invention from different geographical locations by a group of people is termed as Collaborative Invention Mining. It has been presented here in focusing on articulation and segmentation of a raw idea through an exhaustive matrix containing 48 cells in it called as an Idea Detailing Tree (IDT) and ratifying systematically and iteratively maturing towards an invention that can be patentable. The IDT is figured based on three dimensions such as Category, Area and Characteristics. Here Category represents the process of widening the scope of the ideation data, Area defines the lengthening the coverage and Characteristics defines the deepening the sustainability of the invented idea. The collaboration is achieved through a systematic phase as Storm phase Form phase Norm phase Compose phase representing the hierarchical maturity level if the ideation data which is to be converted into the ideation object through the above mentioned approaches. Evaluating the score of the IDT matrix positions and manipulates the cell pattern cell pattern. A system implemented application to exercise CIM is integrated with existing systems for invention management for administration, valuation and portfolio analysis.

4.2 Background

The process of strengthening and maturing a raw idea, i.e., an ideation data through a collaborative interaction by a group of experts from different geographical locations is known as invention mining. The mere purpose of such process is to frame questionnaire to increase capability to widen, lengthen and to deepen the thought and hence maturing it and finally position it as a patentable invention in a selected jurisdiction. In this way an ideation data is made to be emergent sharpened resulting in the establishment of an invention community in an enterprise.

A focused program coupled with management commitment to encourage Inventions (and Innovations) is essential for survival of any business. The outputs need to be protected and monetized through a mature IPR strategy. Collaborative Invention Mining is the process of transforming A Concept or An Idea in an enterprise through a collaborative deliberation into a sustainable invention, applying the EA3 Business Principle. The objective of such a process is to impart various dimensional rigor across category dimension, area coverage and business characteristic to widen, lengthen and deepen an idea, thereby further maturing/composing it as a sustainable, patentable invention. The process in turn, facilitates the emergence of new inventors, sharpens the skill towards inventions and institutionalizes the culture of invention in an enterprise.

To have significant value in the world of intellectual property, an ideation object must be matured in the context of technical, business, regulatory and socio-economics which can be carried forward as Intellectual property creation . This may create a focussed inventors portfolio I a community where ideation data are more resilient and aligned to a business footprint of an enterprise.

4.3 Objectives of Collaborative Invention Mining

Following are the objectives of designing CIM Module:

- To automate the Collaborative Invention Mining activities/process
- To set up an organization wide Inventors Community.
- To institutionalize the culture of sustainability, patentable invention/innovation.
- To create a collaborative platform for knowledge sharing for an Enterprise.
- To map it directly with the TCS Valuation Module for the predictive estimation of Non-linear in revenue.
- To Multi facet Usage of the Template and Process for Portfolio Analysis/ Landscaping, Gap Analysis, FTO Analysis.
- To enable multi-view Capability of the Framework: Stakeholders across all domains of IP .

4.4 System users and Stakeholders

Role	*Description*
Inventor	TCS employee New/existing Inventor
Moderator	TCS employee Member of Inventor community/CIG Member
NIU Hat, PTMS Hat, EA3 Hat	TCS employee Member of Inventor community/CIG Member
Prior Art Analyst	TCS employee – CIG Member/TCS Search Team
Technical Writer	TCS employee Inventor/ Member of Inventor community
Claims Analyst	TCS employee Inventor, CIG Member

Table 4.1: System users and Stakeholders for CIM

4.5 Application development methodology

The aforementioned module (CIM) in developed and integrated with the existing by following the following software development phases.

4.5.1 Business Requirement Analysis and Design

This phase of application development strategy contains the details regarding the requirements from the business point of view. This includes the client or the customer of the application as the TCS Corporate Intellectual Property Rights (IPR) Group who is also the owner of the application. In this phase the Business requirement of the system is analyzed and the documented, which is termed as BRD. This document records hereby the business requirement specifications of the Collaborative Invention Mining Module (CIM). The requirements are specified with the help of a set of Use Cases, and other UML diagrams.Here the use case of the application is discussed as follows.

In this phase several diagrams are drawn according to the BRD which will help in the coding phase of the system. Here, the diagrams are drawn and represented

for several purposes. The use case diagram of the system represents the various use cases carried out during the development of the system. The following diagram represents the different use case. Here the use case diagram represents the different

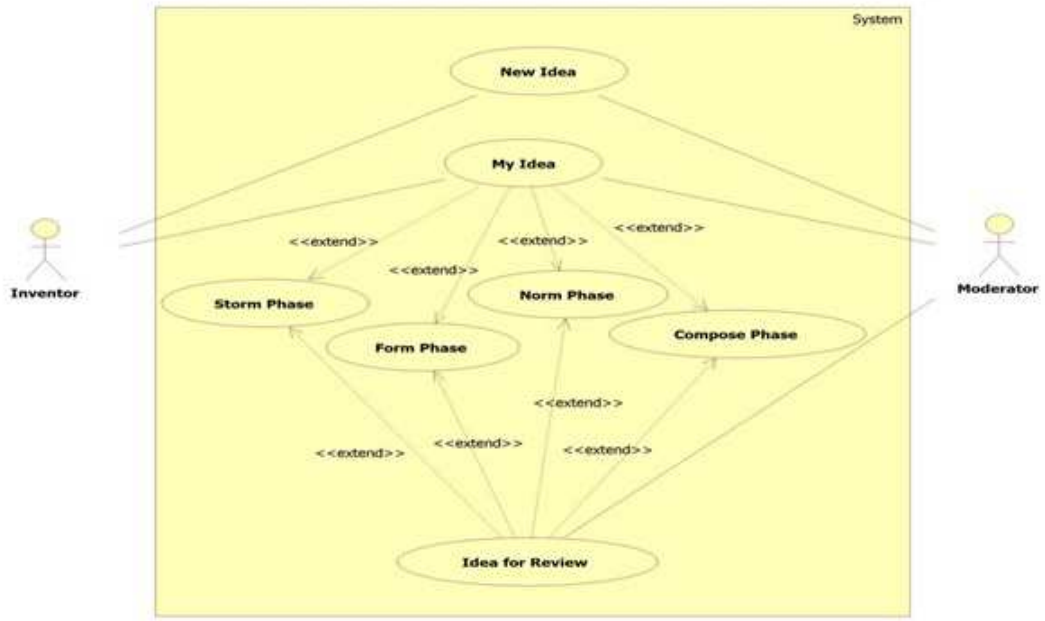


Figure 4.1: Use Case Diagram

functionalities of the application being developed. This is also represented in the tabular form out of which some are shown as below.

After that the behavioral diagrams are drawn that contains the sequence diagram, activity diagram, collaboration diagram and the state machine diagram. Some of these diagrams are depicted for one functionality from each section in the following.

CIM Home Page		
Use case summary	The user lands on the Home page of CIM	
Actors	Inventor (First named Inventor, Co-inventors), Moderator, PAA, Claims Analyst, Technical Writer, NIU Hat/PTMS Hat/EA3 Hat	
Preconditions	**Data**	**System**
	Idea details (Title, keywords, text, diagram, template, presentation, algorithm, etc.)	• The user should have logged in • The user should be authorized to search idea details.
Triggering Event	Initiated by the system	

The main flow of events	
Action	**System Response**
1. The user selects CIM to create, search and review Ideas.	The system opens the landing page, with "New Idea", "My Ideas" button. "Ideas for review" button is visible in case of a Reviewer logged in. In case of "My Ideas", Active and Closed count to be displayed by the system. Active is nothing but, Ideas those Status are not yet closed. In case of "Ideas for Review", For Review count to be displayed, which is number of Ideas for review. Integration of CIM to existing IPRMS to be finalized.
2. User clicks on "New Idea" button.	The system opens Idea Sharing Template (IST) for creation of a new Idea.
1. User clicks on "My Ideas" button. 2. Reviewer clicks on "Ideas for Review" button.	The system opens the Search Result Page. "My Ideas" are ideas created or involved by the Inventor. "Ideas for Review" are ideas pending for review with a Reviewer.
3. The user should able to use the Tools (Settings, Participants, Notepad, Change View, Documents)	▲ The participants - System should navigate to Add participant screen. ▲ The notepad – system should open a Text Editor, which will store the last edited data. ▲ Change View (Matrix view/Pad View) – System will keep it disabled till Parking Lot. ▲ Documents – System allows the user to view and upload the Prior Art Docs to different Phases of the cycle with some mandatory input fields (TCS Internal, Paid Source, Patented Literature) ▲ Settings – will be in a disabled condition.

Variations	
Action	**System Response**
4. None	None

Exceptions	
Action	**System Response**
None	None

Supporting information	

Post-conditions	**Data**	**System**
	None	None

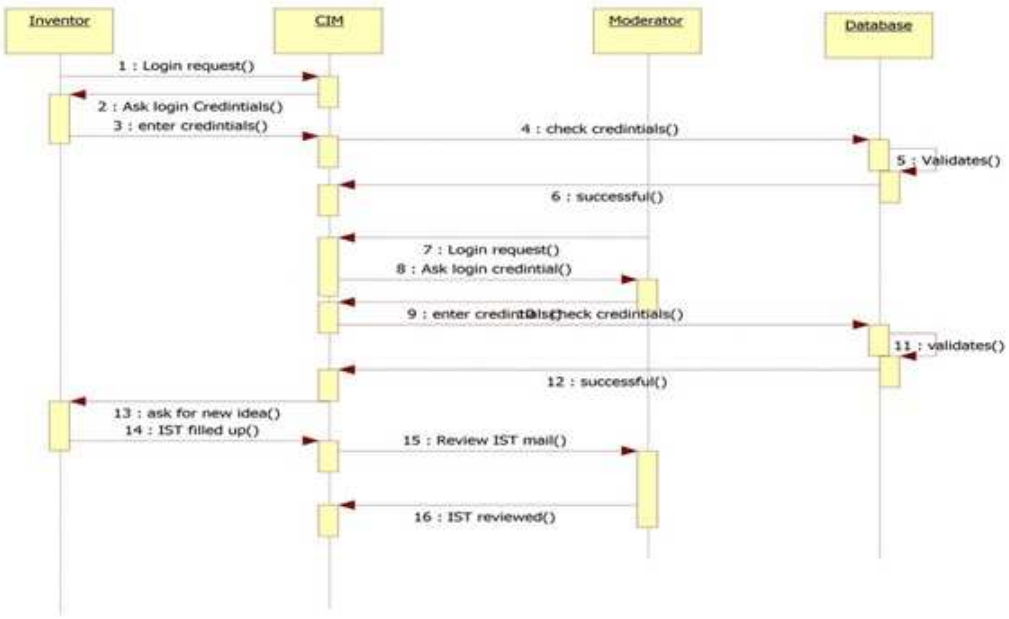


Figure 4.3: Sequence Diagram for IST Review

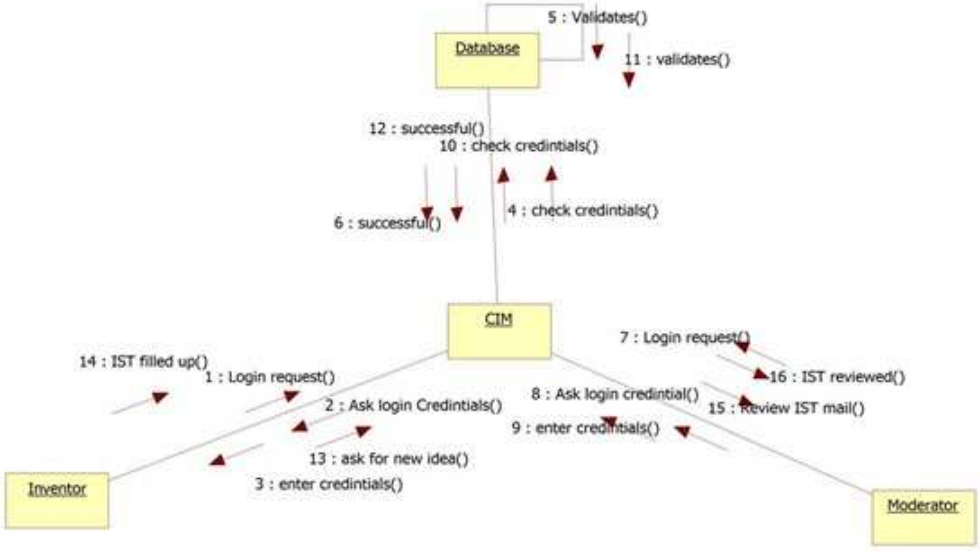


Figure 4.4: Corresponding Collaboration Diagram

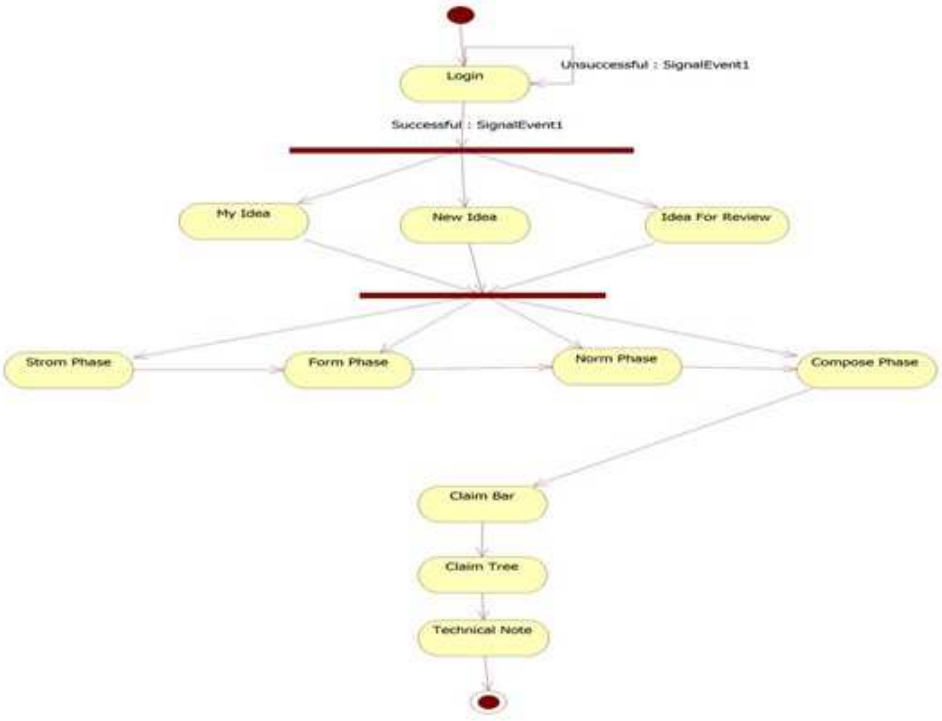


Figure 4.5: Activity Diagram

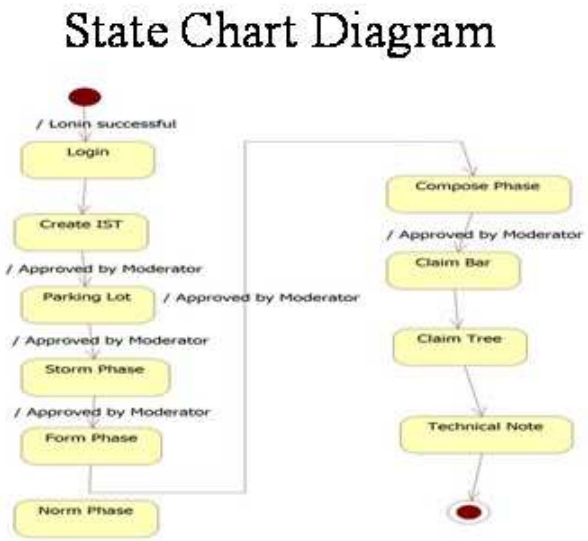


Figure 4.6: State Chart Diagram

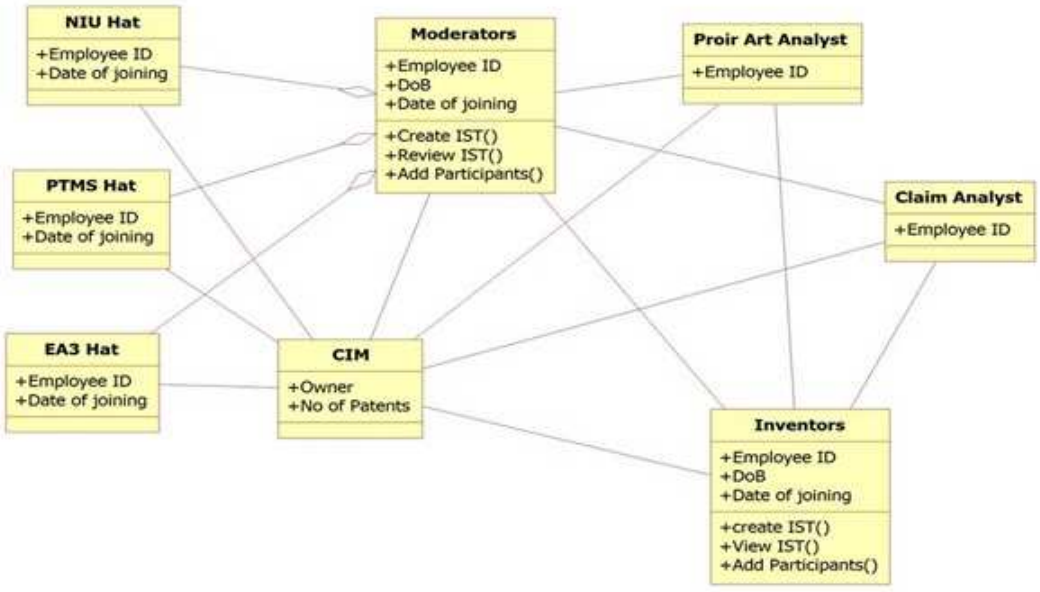


Figure 4.7: Class Diagram

4.5.2 Coding and Implementation Phase

In this phase the code has been developed in FLEX 3.0 technology for UI design and for back-end, the code is developed in Java. For FLEX Caraingorm framework is used to interact between the back end and front end through Data Transfer Object (DTO). Basically, we were involved in the development of the front end of the application.In this phase, the front-end of the application been developed using Adobe Flex 3.0 technology. Some of the screen shots of the application user interface part are depicted as follows.

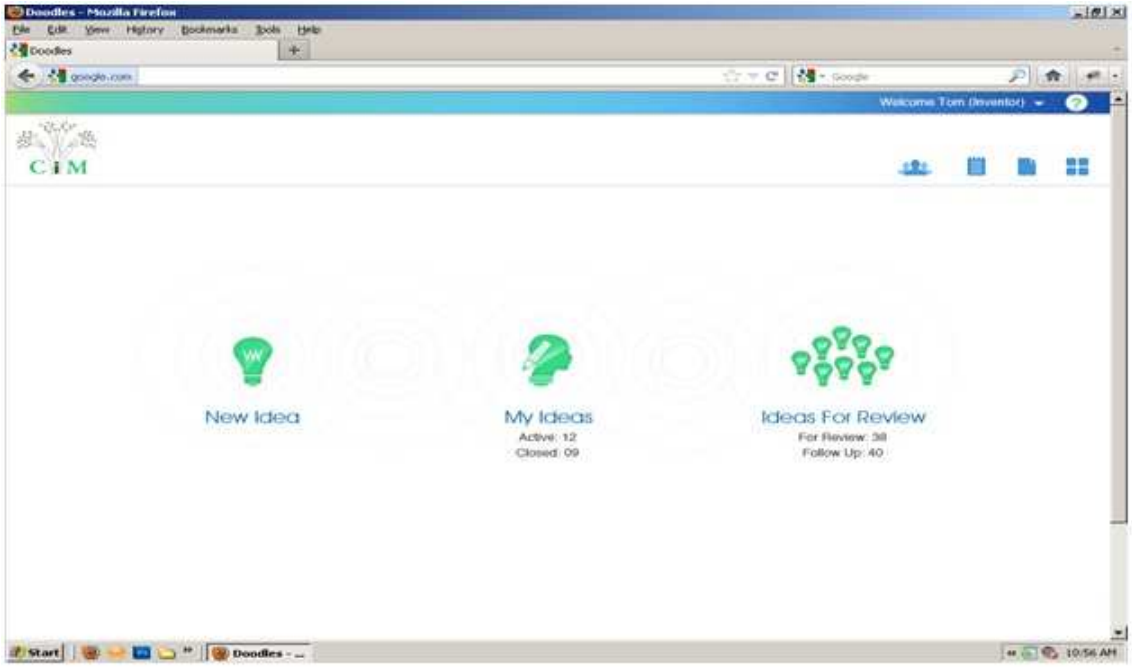


Figure 4.8: CIM Landing Page

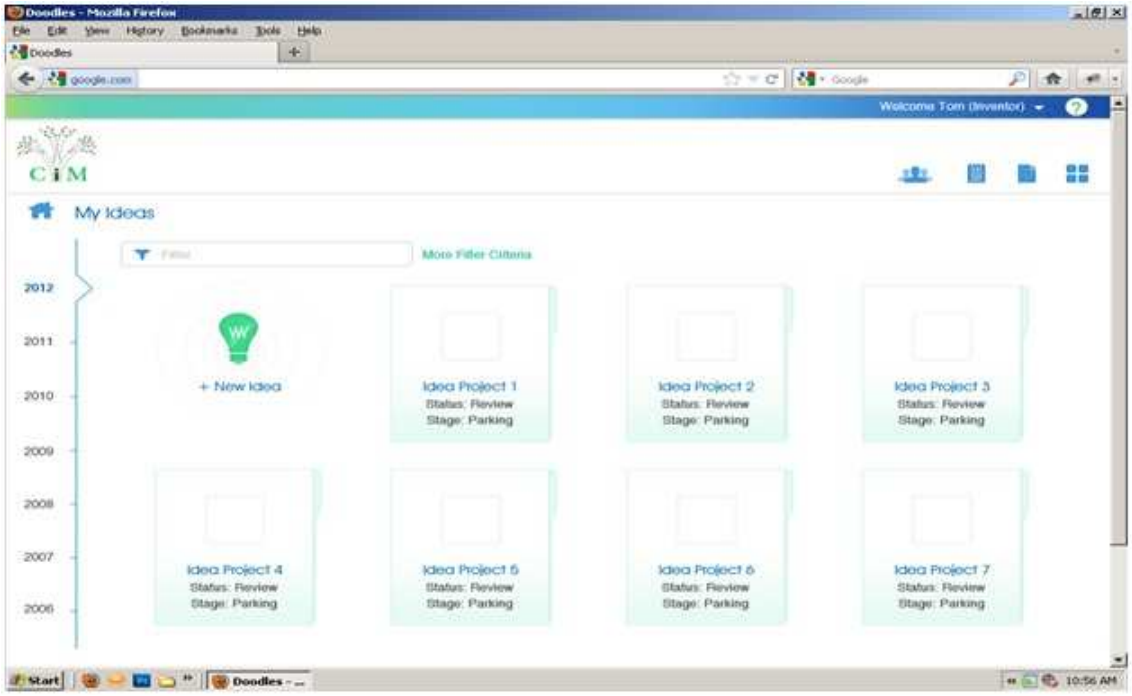


Figure 4.9: My Ideas

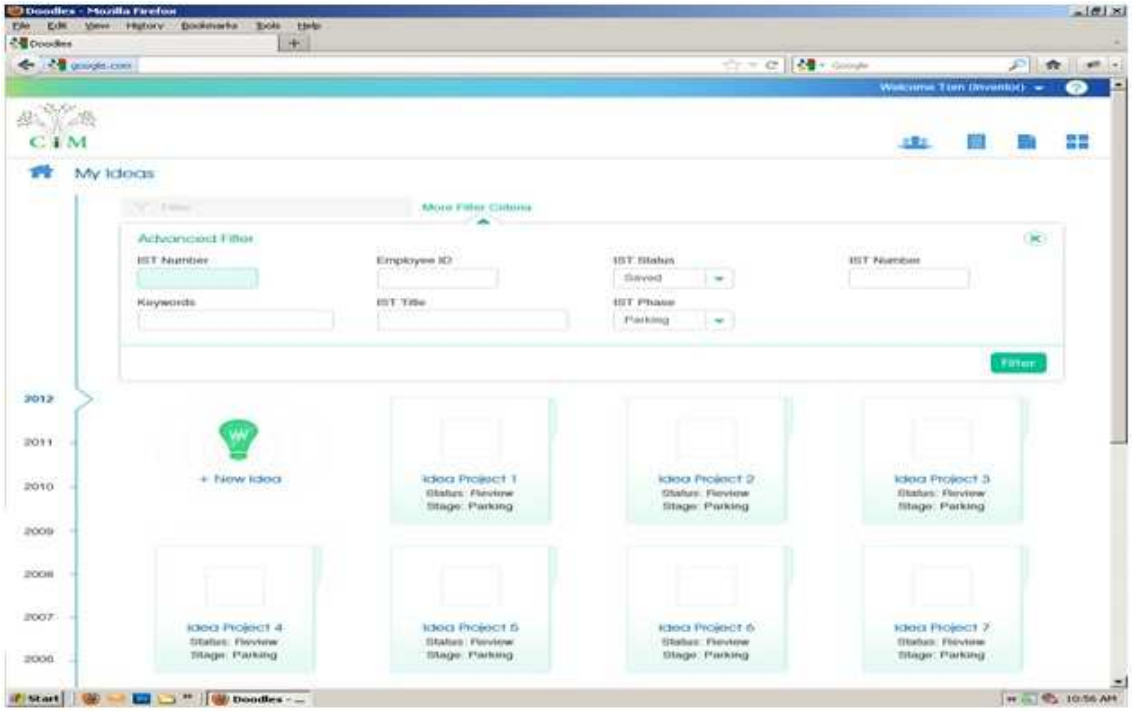


Figure 4.10: Advance Filter

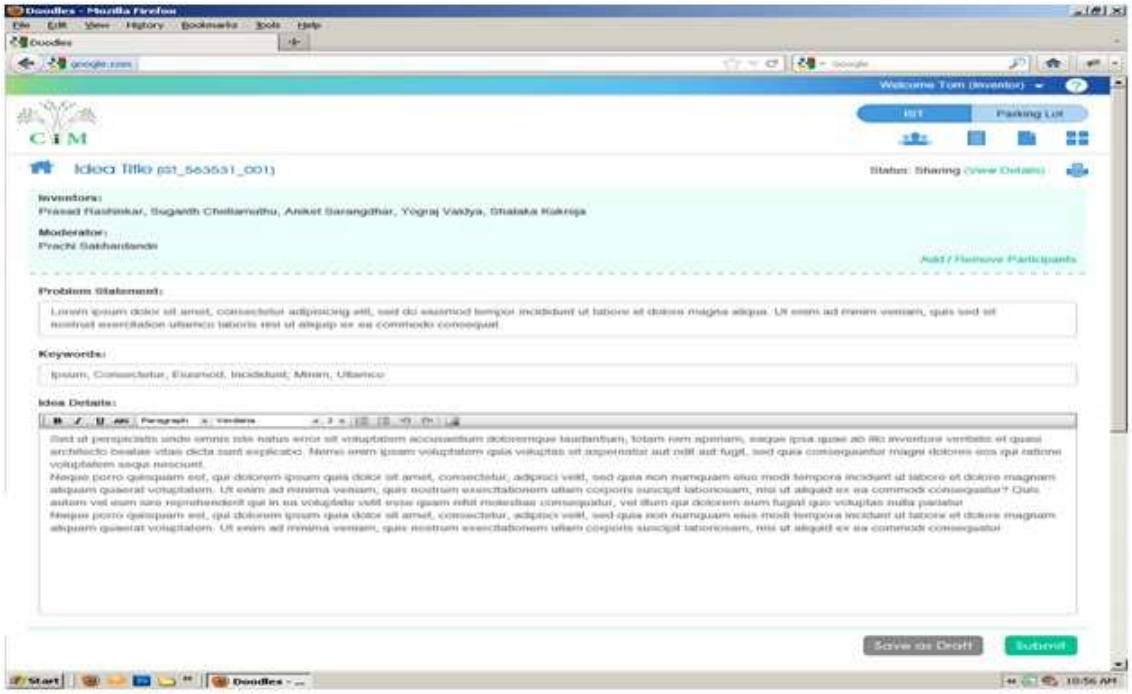


Figure 4.11: Idea Sharing Template

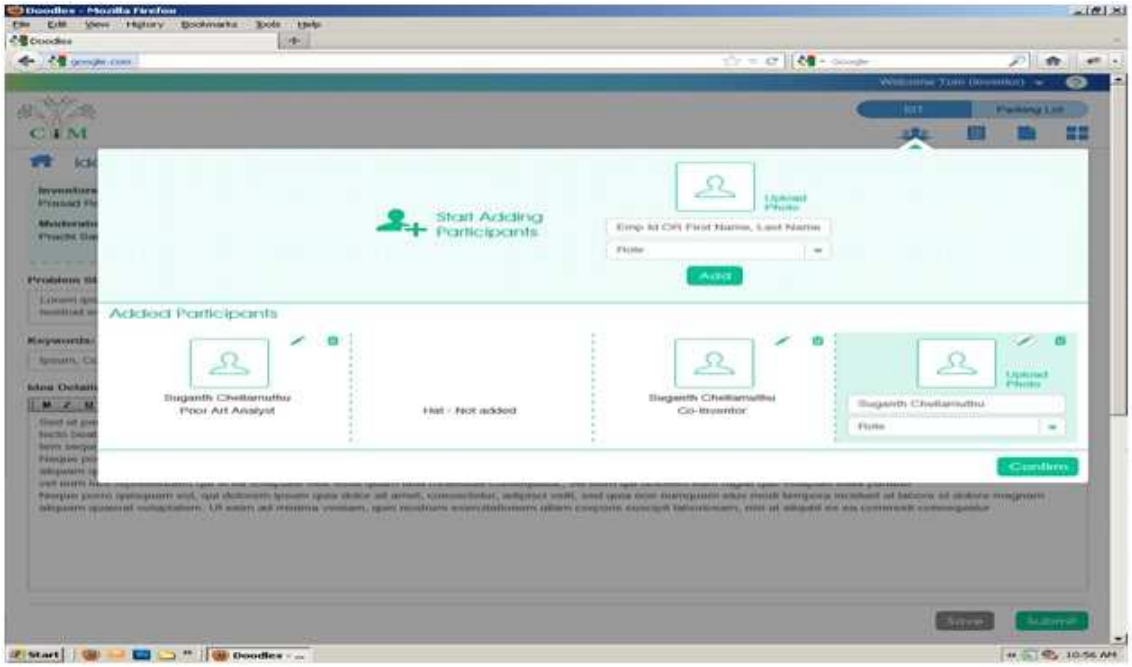


Figure 4.12: Add Participants

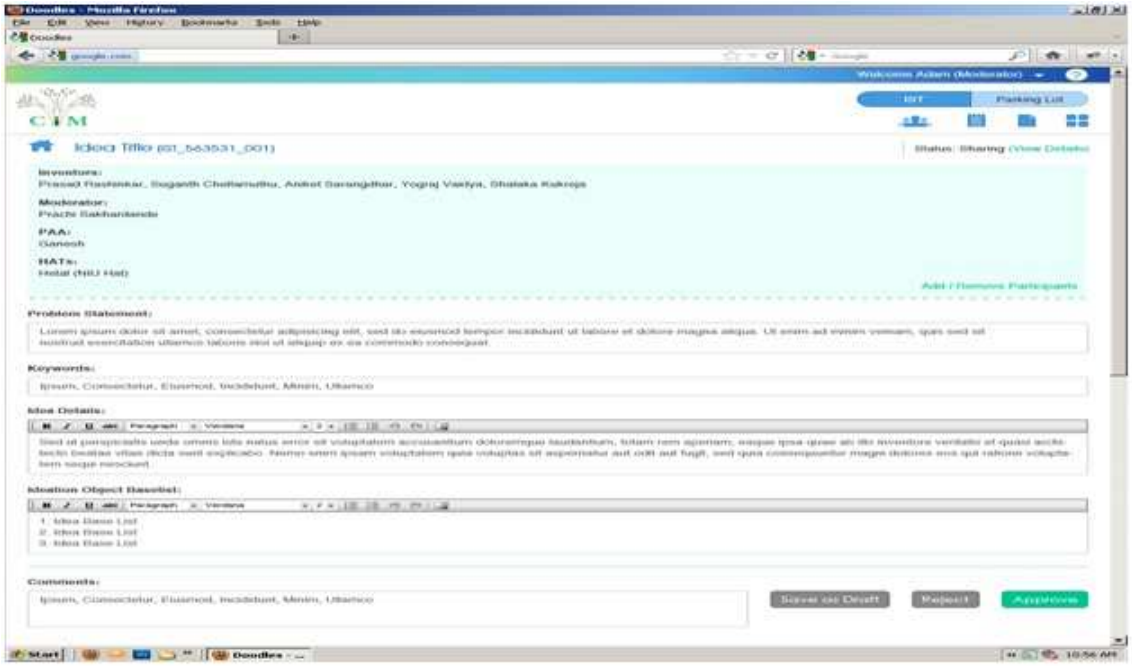


Figure 4.13: Idea Sharing Template Review

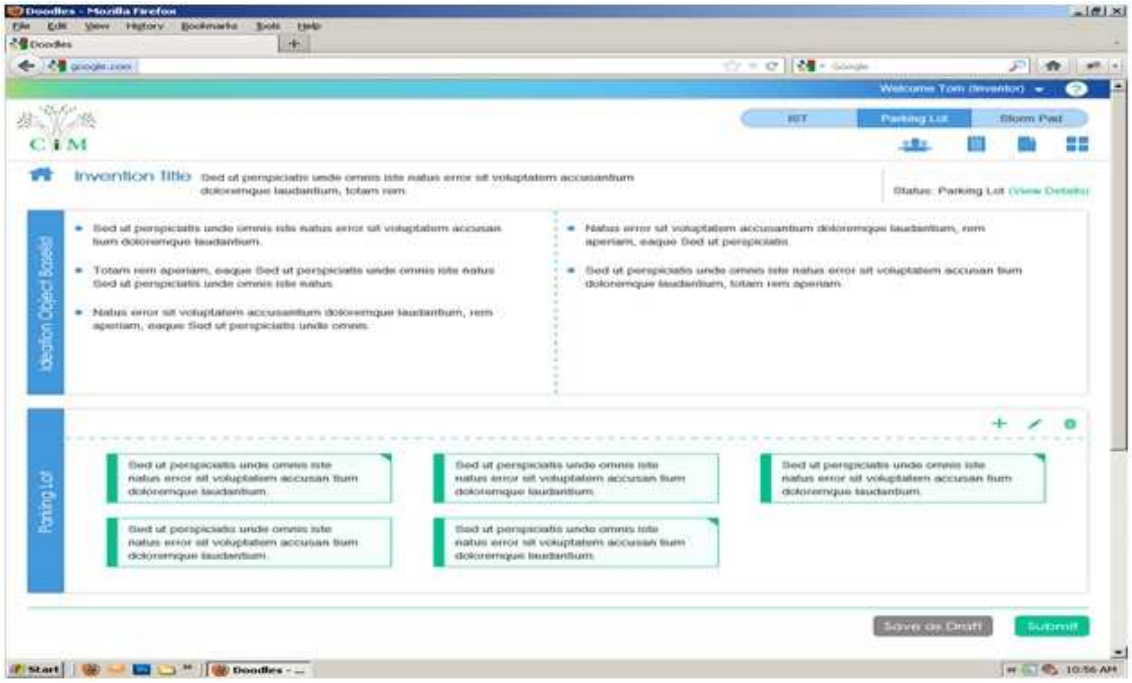


Figure 4.14: Parking Lot

Figure 4.15: Storm Phase

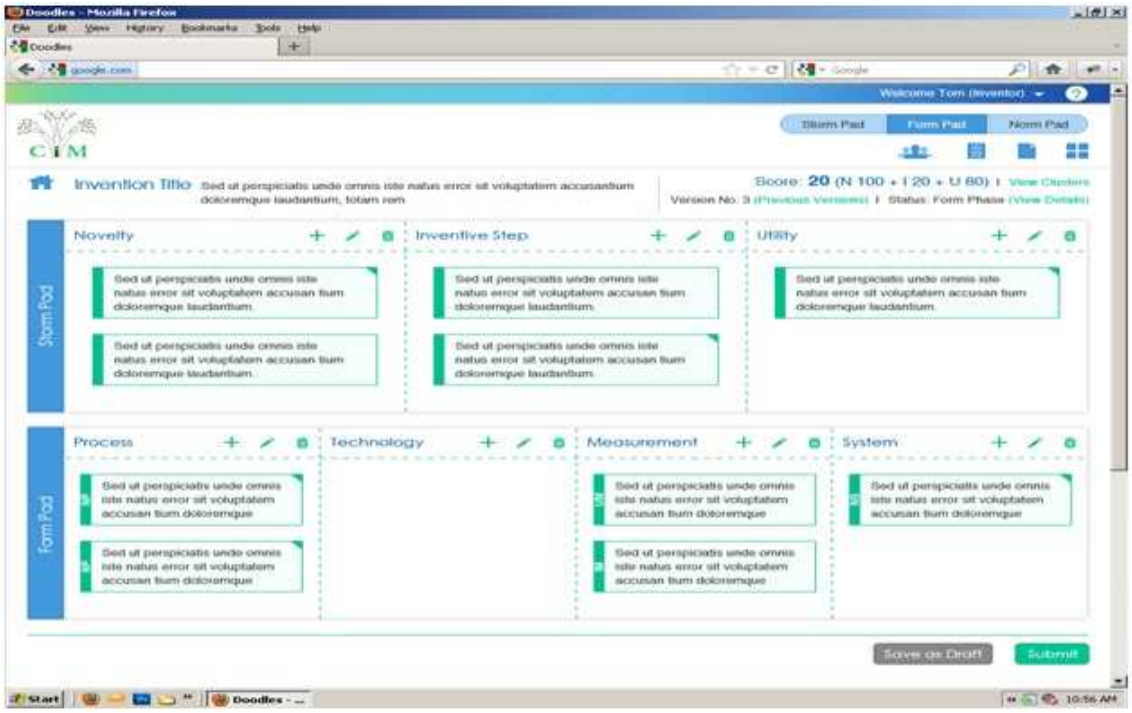


Figure 4.16: Form Phase

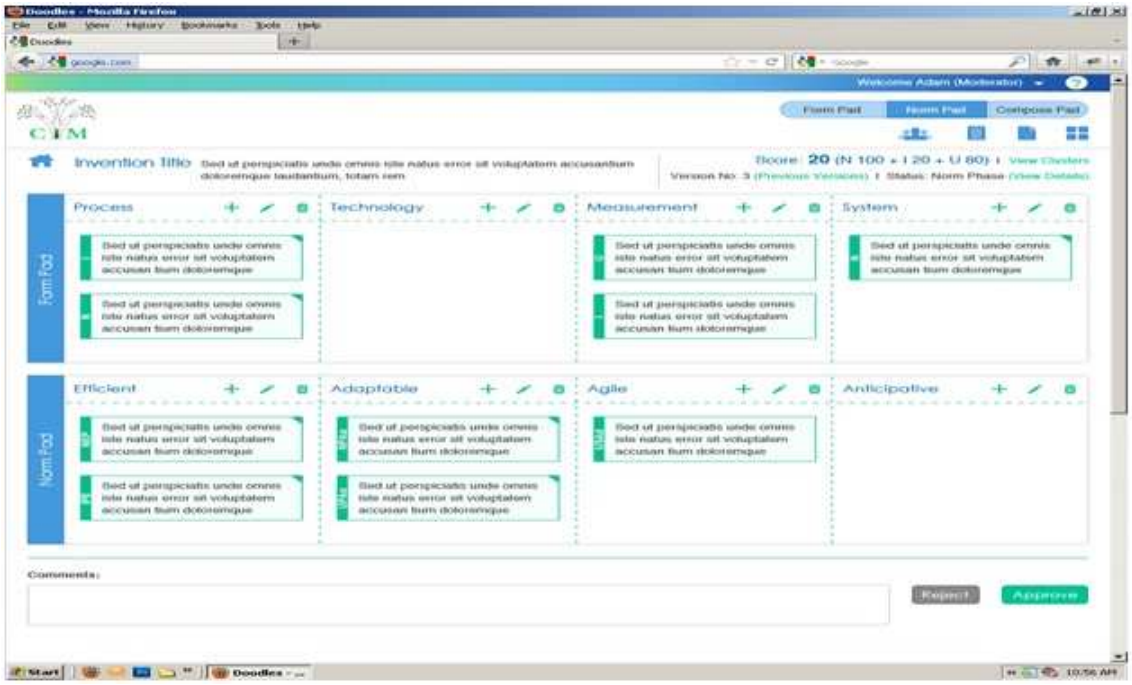


Figure 4.17: Norm Phase

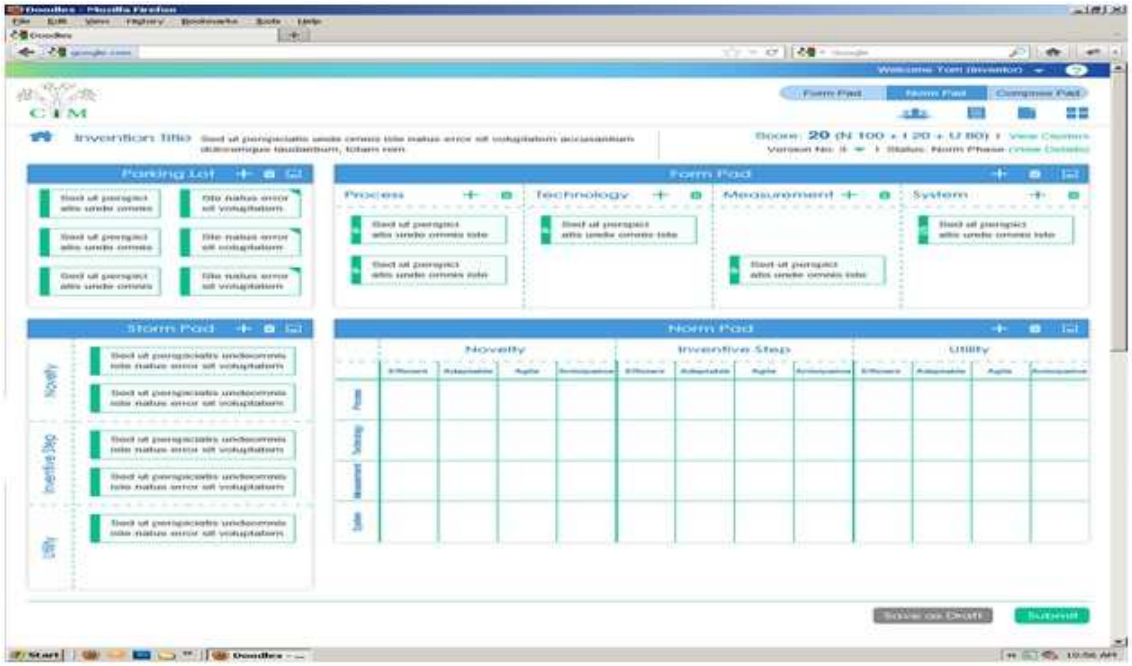


Figure 4.18: Matrix View

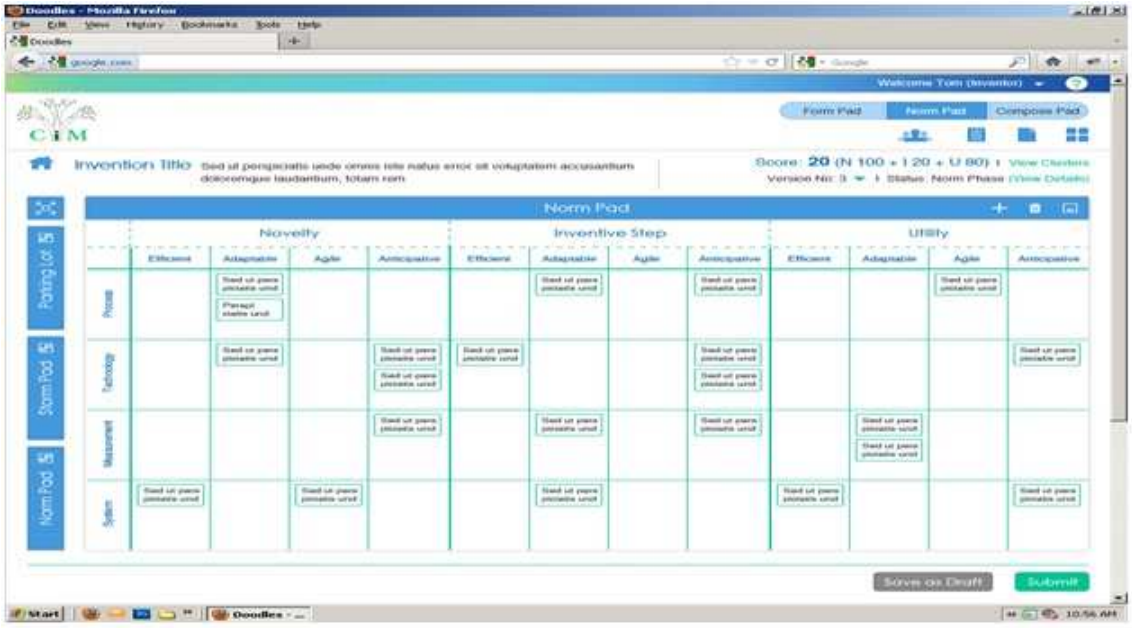


Figure 4.19: Norm Pad Matrix View

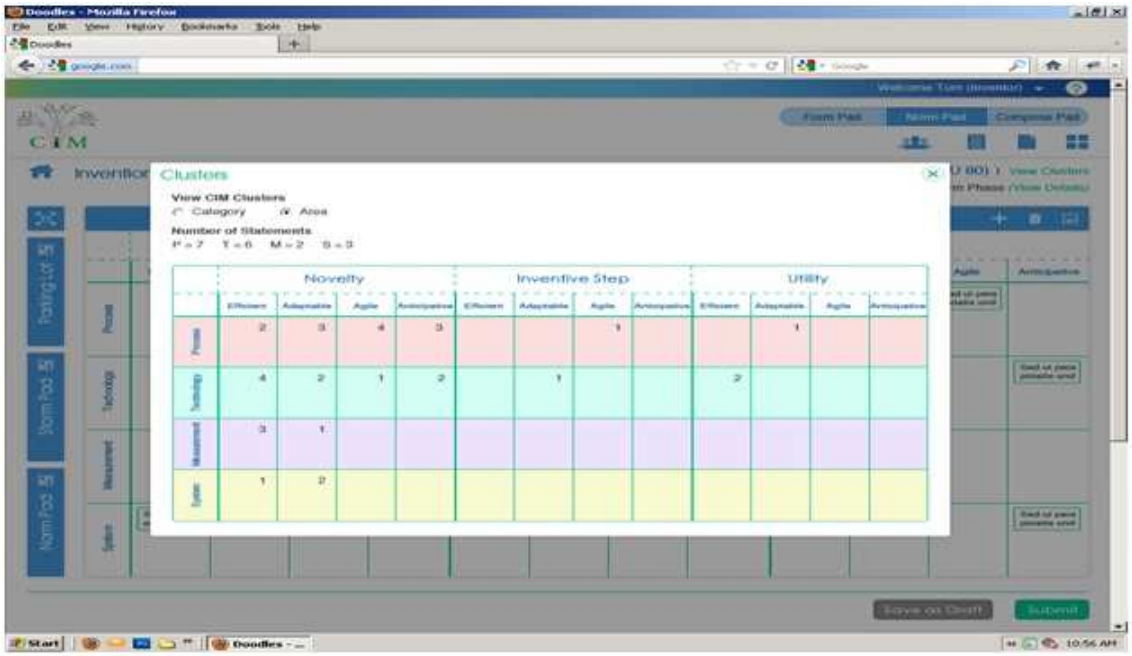


Figure 4.20: Cluster by Area

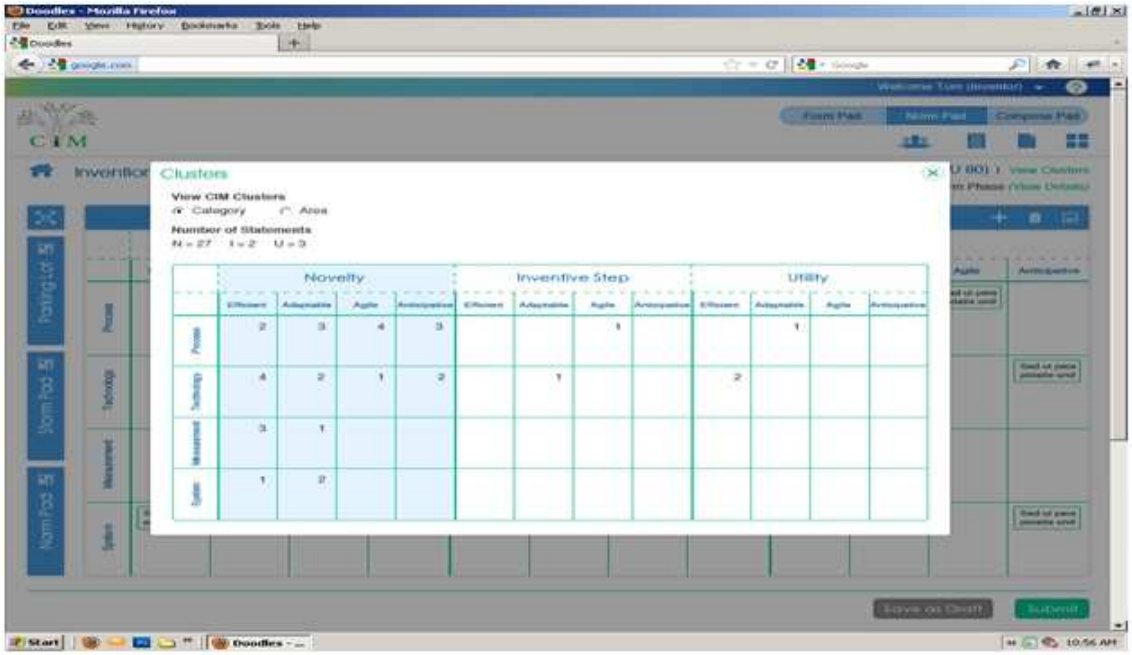


Figure 4.21: Cluster By Category

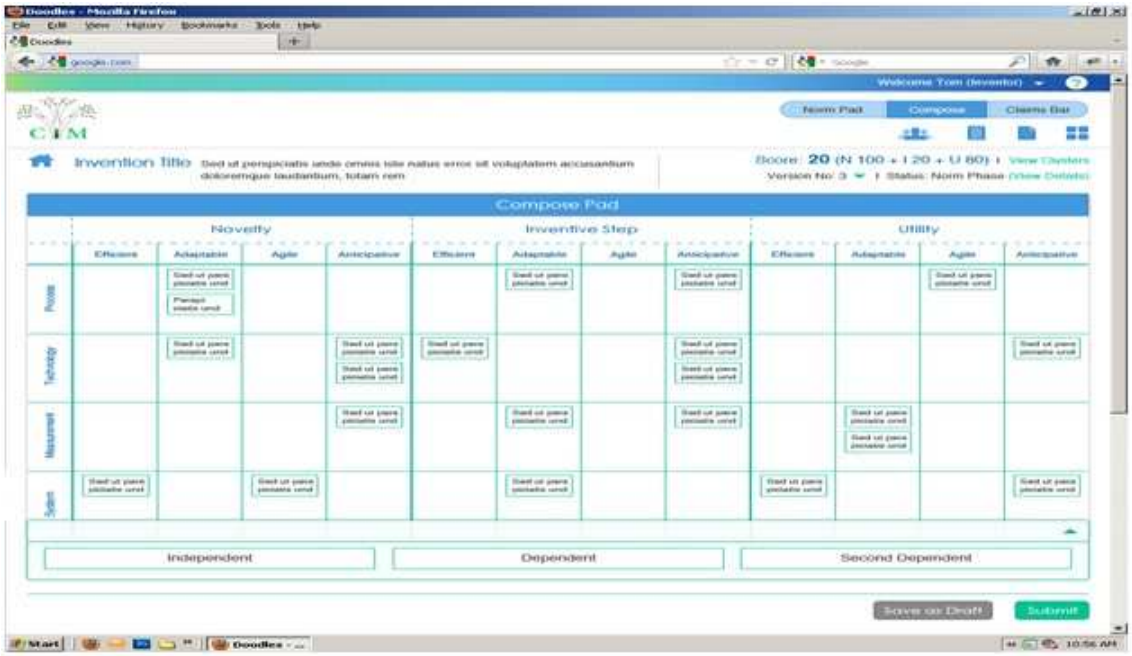


Figure 4.22: Compose Phase

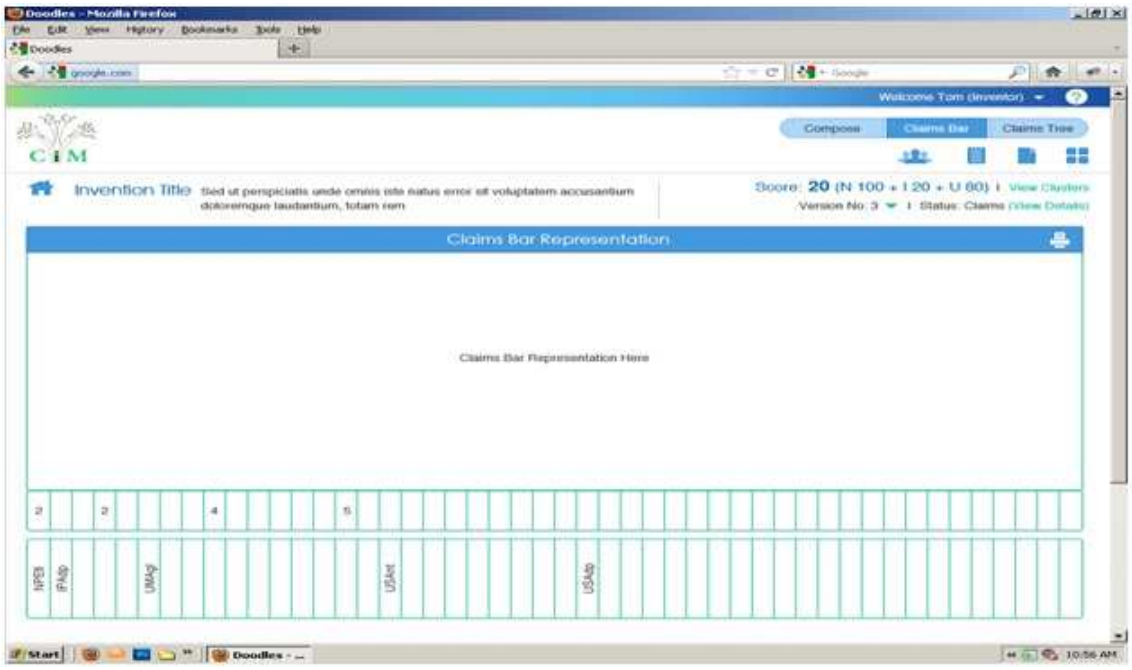


Figure 4.23: Claim Bar Representation

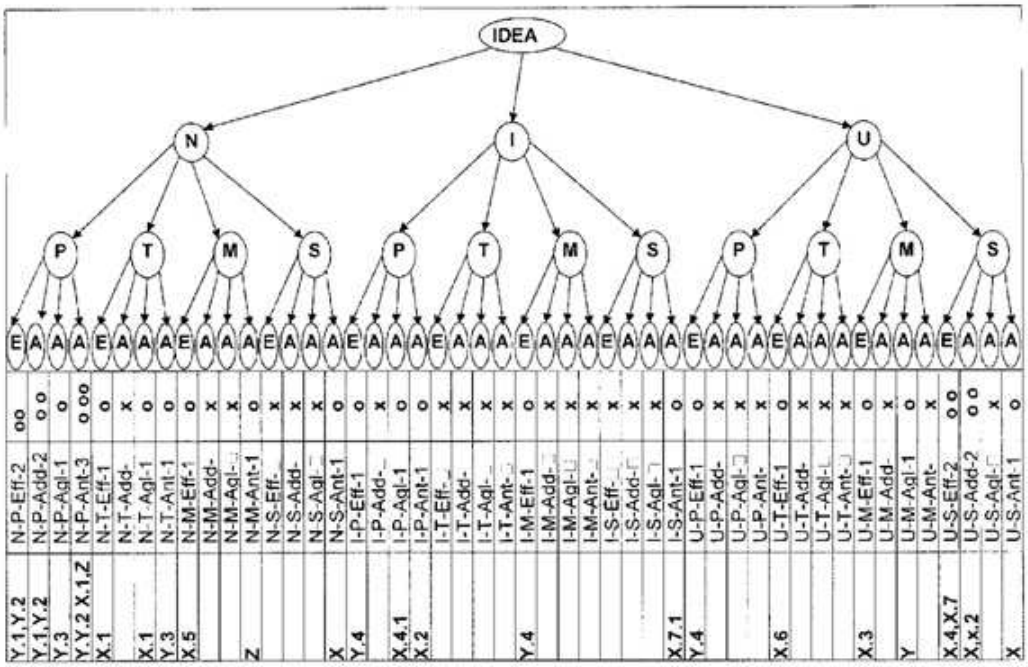


Figure 4.24: Claim Tree Representation

4.5.3 Testing Phase

In this phase, initially the unit test cases are written according to the functionalities and these are uploaded in the Application Life Cycle manager (ALM)-Testing tool. Then the TCs are mapped to the requirements with the help of the ALM tool. Simultaneously the system TCs are written by the testing team, according to the Business Requirement Documents (BRD). Then the system TCs are uploaded in the ALM tool for execution. 3600 TCs are written and uploaded in the ALM tool for execution. The Test case has special format for its execution.

Chapter 5

Test Case Generation Using Activity Diagram

5.1 Basic Concepts

Before entering into our thesis, we should have some basic idea regarding generation og test cases, some testing terminologies, etc. In this chapter, we discuss the basic definitions and terminologies, on which our research is based.

Test Case: A test case is a set of trios such as, set of inputs, processing conditions, and expected outputs which is developed to achieve a particular objective, such as exercising a specific scenario, a particular scenario sequence or to verify compliance with a particular argument.

Test Adequacy Criterion: As we realize that the dependability of a software application accomplished by testing the application which makes the software bug free. Be that as it may, when to quit testing is still unanswerable. Contingent upon a few demands and due dates, the testing process typically halted. A paramount standard of those is a test adequacy criteria which demonstrate the predicate sufficiency. Transition coverage, branch coverage, path coverage and activity coverage are few of such test sufficiency criteria, though we have utilized just

transition coverage and path coverage for our work. A test sufficiency foundation helps in deciding test targets that are to be attained while performing a particular software testing. Case in point, path coverage obliges that each path in a system under test is to be practised by at least once.

5.1.1 Brief notes on Model Based Testing

To describe the behaviors of a system by designing its model has contributed a major advantage to the testing team of an enterprise. In many ways, model can be used throughout the development life cycle with improvement in quality of specification, analysis of reliability and generate test cases.

A model is used to describe the concurrent behavior of a system as a result of which it can be used to represent the system hence can be used for system testing. This model can also be used to analyse its quality, robustness et al [12].

Model-based testing has turned into another and advancing method to create a suite of test cases from the software requirement. The testers utilize this methodology and concentrate on an data model and create framework as opposed to handcrafting unique tests. Numerous a little research has been done, how combinatorial test derivation process permit the testers to attain expansive scope of the data area with a little number of tests [13]. We have led a few generally extensive application in which we connected these procedures to frameworks with a large number of lines of code. Given the intricacy of testing, the model based testing methodology was utilized within conjunction with test automation outfits and as no vast exact study has been led to measure adequacy of this new approach, we cover our experience with creating models and systems in backing of model-based testing [14].

5.1.2 Overview of UML Diagrams

Unified Modeling Language (UML) is a semi-final visual modeling language, which is a collective approach of trio James Rumbaugh (Object Management Technology), Grady Booch (Boochs Methodology) and Ivar Jacobson

(Object-Oriented Software Engineering). It was adopted as a de facto standard for modeling software systems by the OMG in 1997 [15]. Of late, popularity of UML models in academic and industrial levels is attracting the focus of researchers for test case generation in the context of model based testing. The UML diagrams generally describe the different views of the system. In one view it describes the users', In second view, it describes the structural view. In third, it describes the implementation view and in fourth, it describe the behavioral view. In fifth, it describes the Environmental View.

An automatic approach for test case generation will be futile if we proceed at the end of the development phase. So, automatic test case generation using design document (or system specification or model) is more reasonable.

Activity Diagram

An alternative imperative outline in UML to portray the concurrent behaviors of the framework is the activity diagram. Activity diagram is an alternate imperative diagram in UML to portray dynamic aspects of the framework.

Activity diagram is fundamentally a flow chart to speak to the flow from one activity to another activity. The activity might be portrayed as an operation of the framework.

This flow could be branched, concurrent or sequential. Activity diagram manages numerous types of flow control by utilizing distinctive components like decision node, fork, join and so on.

Purpose of Activity Diagram

The essential purposes of activity diagram are like other four diagrams. It catches the dynamic behaviors of the framework. Other four diagrams are utilized to show the message stream starting with one item, then onto the next, yet the activity diagram is utilized to show message stream starting with one action then onto the next.

Activity is a specific operation of the framework. Activity diagrams are

not just utilized for picturing dynamic behaviors of a framework, yet they are additionally used to develop the executable framework by utilizing forward and reverse engineering. The main limitation of this diagram is that it does not contain the message in the path.

It doesn't demonstrate any message path starting with one activity, then onto the next. Activity diagram is sooner or later acknowledged as the flow chart. Despite the fact that the diagram resembles a flow diagram, yet it is most certainly not. It indicates diverse flow like single, branched, parallel, concurrent et al.

Fundamentally, the basic purposes of the activity diagram of an application or a framework can be best demonstrated as:

- Draws the activity flow of the application or framework.
- Describes the sequential path from one activity to another.
- Describes the branched, concurrent, parallel flow of the application.

How to draw Activity Diagram?

Activity diagrams are for the most part utilized as a flow diagram is comprised of activities performed by the framework. However activity diagrams are not precisely a flow chart as they have some extra abilities. These extra capacities incorporate parallel flow, branching, swimlane and so on.

Before drawing an activity diagram, we must have a reasonable idea about the components utilized within activity diagram. The fundamental component of an activity diagram is the activity itself. An activity is a unit of work performed by the framework. In the wake of recognizing the activities we have to see how they are connected with stipulations and conditions.

So before drawing an activity diagram, we should identify the following elements:

- Activities
- Association

- Conditions
- Constraints

Once the aforementioned parameters are distinguished we have to make a mental design of the whole flow. This mental design is then converted into an activity diagram. Here the activities are represented as follows, which is followed by the transition components of activity diagram.

Symbols	Names
	Swim lane
	Initial node
	Final node
	Activity
	Control flow

Figure 5.1: Activity Components

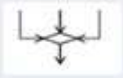
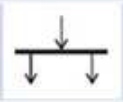
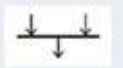
Symbols	Names
	Decision node
	Merge node
	Fork
	Join

Figure 5.2: Trnasition Components

Where, the business process is shown by the activity diagram with a sequence of activities. It can also be used to show the software as a work flow through a sequence of activities. Some of the fundamental elements of activity diagram and their use are described below.

- Swimlanes: It is used to configure the activities into different areas corresponding to different components or business requirements that can be used to perform an action.
- Initial Node: The node from which the control starts after an activity is called.
- Final Node: This node indicates that the activity is completed.
- Decision Node: The decision node indicates decision control from which multiple outgoing can be possible with a single input.
- Merge Node: Merge node combines multiple incomes and produces a single out put.

5.2 Implementation and Result

In this section, we discuss the approach by which we have implemented our research work. We have considered the application termed as Collaborative Invention Mining (CIM) implemented in Tata Consultancy Services, Bhubaneswar as the primary system based on which we implement the automatic test case generation from its activity diagram. The step by step approach defined below represents and outlook of the work.

- Activity diagram using IBM RSA (Rational software Architect).
- Generate XMI code from the activity diagram using RSA.
- Parse the XMI code using Java code to generate the corresponding test scenarios.

- Traced out the paths using GVEdit software resulting in a graph containing the test paths.

The activity diagram represents the co-ordination among the different activities being carried out during the process of implementation of Collaborative Invention Mining. Here is the activity diagram for the CIM.

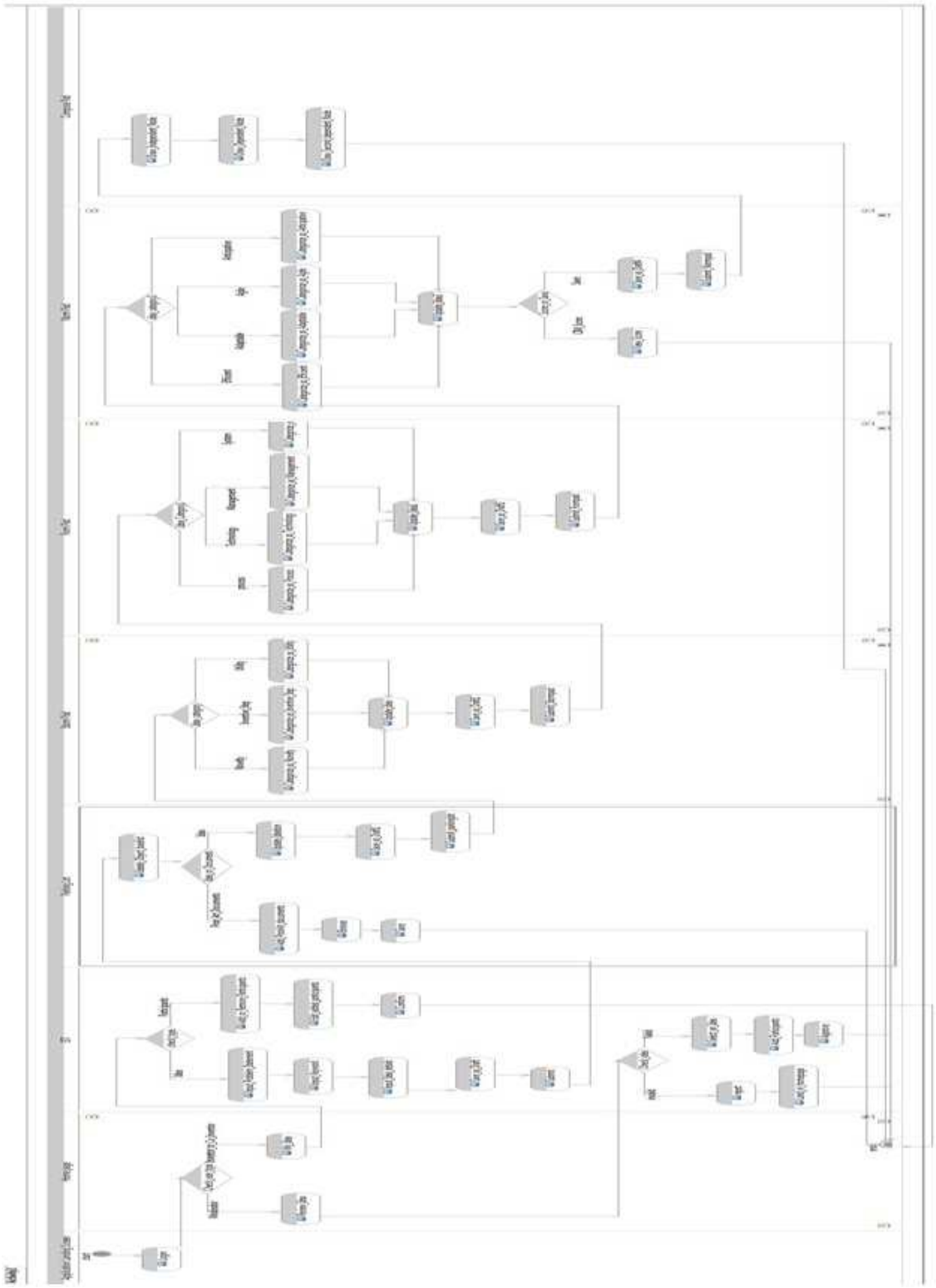


Figure 5.3: Activity Diagram for CIM

5.2.1 XMI Generation

The XMI stands for XML Metadata Interchange is a standard for exchanging metadata information through XML (Extensible Markup Language). XMI is developed by the Object Management Group (OMG). It consists of the textual representation of the activity diagram that has been drawn in RSA. The corresponding XMI of the above diagram is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:uml="http://schema.omg.org/spec/UML/2.1.1" xsi:schemaLocation="http://schema.omg.org/spec/UML/2.1.1 http://www.eclipse.org/uml2/2.0.0/UML" xmi:id="_sDj_cLDtEeOsFO_98BHCsg"
name="Blank Model">
  <packageImport xmi:type="uml:PackageImport" xmi:id="_sDj_cbDtEeOsFO_98BHCsg">
    <importedPackage xmi:type="uml:Model" href="http://schema.omg.org/spec/UML/2.1.1/uml.xml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:Activity" xmi:id="_sDj_crDtEeOsFO_98BHCsg" name="Activity1">
    <node xmi:type="uml:InitialNode" xmi:id="_sDj_c7DtEeOsFO_98BHCsg" name="start" outgoing="_sDj_rLDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_dLDtEeOsFO_98BHCsg" name="Login" outgoing="_sDj_2bDtEeOsFO_98BHCsg" incoming="_sDj_rLDtEeOsFO_98BHCsg"
inPartition="_sDknELDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_dbDtEeOsFO_98BHCsg" name="My_Idea" outgoing="_sDj_4rDtEeOsFO_98BHCsg" incoming="_sDj_37DtEeOsFO_98BHCsg"
inPartition="_sDknCbDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_drDtEeOsFO_98BHCsg" name="Review_Idea" outgoing="_sDkmxLDtEeOsFO_98BHCsg" incoming="_sDj_3LDtEeOsFO_98BHCsg"
inPartition="_sDknCbDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_d7DtEeOsFO_98BHCsg" name="Input_Problem_Statement" outgoing="_sDj_67DtEeOsFO_98BHCsg" incoming="_sDj_6LDtEeOsFO_98BHCsg"
inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_eLDtEeOsFO_98BHCsg" name="Input_Keyword" outgoing="_sDj_7rDtEeOsFO_98BHCsg" incoming="_sDj_67DtEeOsFO_98BHCsg"
inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_ebDtEeOsFO_98BHCsg" name="Input_Idea_details" outgoing="_sDkmx7DtEeOsFO_98BHCsg" incoming="_sDj_7rDtEeOsFO_98BHCsg"
inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_erDtEeOsFO_98BHCsg" name="Add_or_Remove_Participants" outgoing="_sDj_r7DtEeOsFO_98BHCsg"
incoming="_sDj_5bDtEeOsFO_98BHCsg" inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_e7DtEeOsFO_98BHCsg" name="Edit_added_participants" outgoing="_sDj_srDtEeOsFO_98BHCsg" incoming="_sDj_r7DtEeOsFO_98BHCsg"
inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_fLDtEeOsFO_98BHCsg" name="Confirm" outgoing="_sDknrbDtEeOsFO_98BHCsg" incoming="_sDj_srDtEeOsFO_98BHCsg"
inPartition="_sDknCrDtEeOsFO_98BHCsg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_sDj_fbDtEeOsFO_98BHCsg" name="Save as Draft" outgoing="_sDj_fbDtEeOsFO_98BHCsg" incoming="_sDkmx7DtEeOsFO_98BHCsg"
```


Figure 5.4: A Snap Shot of XMI Code

5.2.2 XMI to Test Case Generation

Here, the conversion of XMI code to test cases is divided into two parts. In the first section, we convert the activity diagram into test paths. For this we use an algorithm called as Activity Path Traversal(APT). And in the second section, we generate the test cases from the test path generated in the first section and for this we use an algorithm called as Test Path Traversal (TPT). Apart from that we validate the generated test path with the corresponding activity diagram by using GraViz Editor which represents the graphical view of the intermediate test

path generated. Finally, we get the test cases.

Activity to test path generation

In this approach we implement APT algorithm which takes the activity diagram as input and produces the test paths as the output. The algorithm is illustrated as below:

Activity Path Traversal (APT) Algorithm

Input :Activity control flow graph (ACFG).

Output :All test paths.

Step 1: push the initial node of the activity control flow.

Step 2: pop the node from the current stack of the graph(ACFG) in the currentstack.

Step 2.1: push it to the display stack.

Step 2.2: if the node popped is not a condition node push the adjacent node of the popped node to the currentstack.

Step 2.3 :if the node popped is the condition node then push it to the conditionstack and push all the adjacent nodes to the currentstack.

Step 2.4: if the node poped is the final node then display the element of the displaystack and pop the element of displaystack until the top of the displaystack is equal to the top of the conditionstack.

step 2.4.1: pop the node from conditionstack

Step 3: repeat above step until the currentstack is empty.

In this way, the test paths are being generated from the Activity Control Follow Graph (ACFG). Here we have generated 100 test paths for the application that we have implemented. Some of the test cases are as follows.

test path 1: start –> Login –> Check_user_type –> My_Idea –> check_type –> Add_or_Remove_Participants –> Edit_added_participants –> Confirm –> end.

test path 2 : start –> Login –> Check_user_type –> My_Idea –> check_type –> Input_Problem_Statement –> Input_Keyword –> Input_Idea_details –> Save_textttas_Draft –> Submit –> Ideate_Object_baselist –> Idea_or_Document –> Update_Ideation –> Save_as_Draft1 –> Submit_parkinglot –> Idea_category –> Categorize_as_Utility –> Update_Idea, Save_as_draft2 –> Submit_Stormpad –> Idea_Category1 –> Categorize_as_System –> Update_Idea1 –> Save_as_Draft3 –> Submit_Formpad –> idea_category3 –> Categorize_as_Anticipative –> Update_Idea3 –> Score_or_Save –> Save_as_Dreft4 –> Submit_Normpad –> Draw_Independent_Node –> Draw_Dependent_Node –> Draw_Second_dependent_Node –> end.

test path 3:start –> Login –> Check_user_type –> My_Idea –> check_type –> Input_Problem_Statement Input_Keyword –> Input_Idea_details –> Save_as_Draft –> Submit –> Ideate_Object_baselist –> Idea_or_Document –> Update_Ideation –> Save_as_Draft1 –> Submit_parkinglot Idea_category Categorize_as_Utility –> Update_Idea –> Save_as_draft2 –> Submit_Stormpad –> Idea_Category1 –> Categorize_as_System –> Update_Idea1 –> Save_as_Draft3 –> Submit_Formpad –> idea_category3 –> Categorize_as_Anticipative –> Update_Idea3 –> Score_or_Save –> View_Score –> end.

test path 15: start –> Login –> Check_user_type –> Categorize_as_Adaptable –> Update_Idea3 –> Score_or_Save –> View_Score –> end.

Test Path Generation

In this algorithm, it takes the above generated test paths as input and generate the corresponding test cases by using following steps.

Test Path Traversal (TPT) Algorithm

Step 1: Traverse the path in sequence.

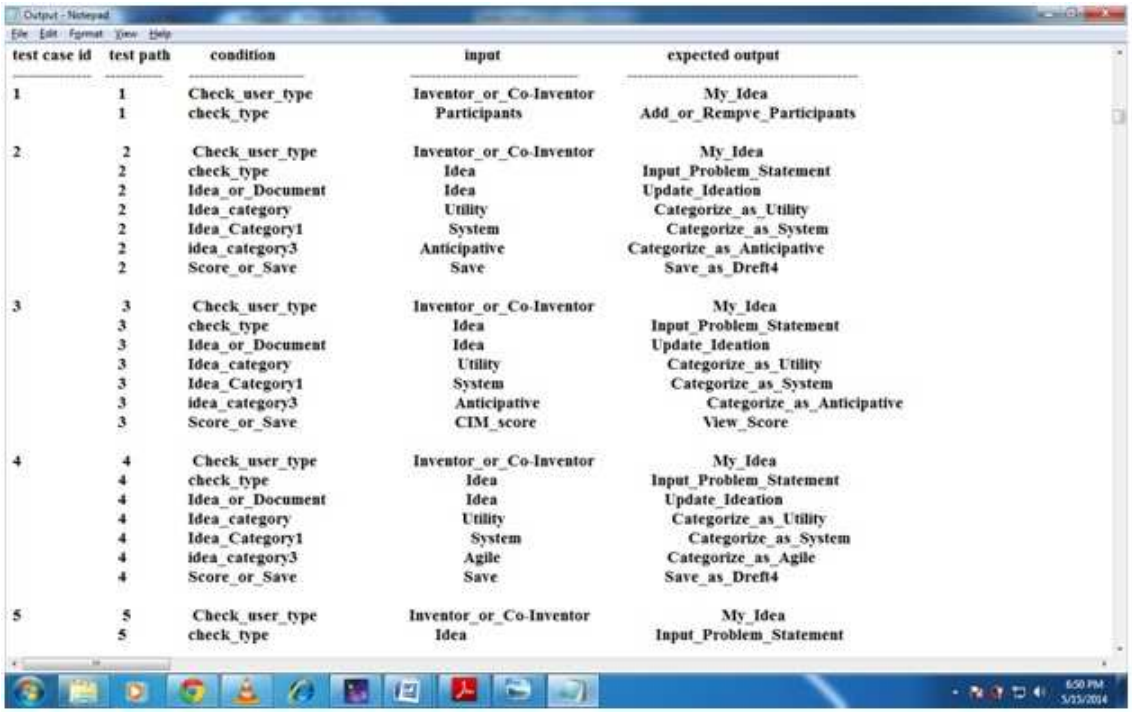
Step 1.1: if the current node is not a condition node then go to next node.

Step 1.2: if the current node is a condition node then generate the test cases where condition node represent the current state and labelled edge represent the input, and next node is expected output.

Step 1.3: if the current node in end node then go to to next path.

Step 2: repeat the above step until all paths are travelled.

In this way, the test cases are being generated from the Test paths generated from the above algorithm. Here we have generated 1944 test cases from 100 test paths for the application that we have implemented. Some of the test cases are as follows.



Output - Notepad

File Edit Format View Help

test case id	test path	condition	input	expected output
1	1	Check_user_type	Inventor_or_Co-Inventor	My_Idea
	1	check_type	Participants	Add_or_Rempve_Participants
2	2	Check_user_type	Inventor_or_Co-Inventor	My_Idea
	2	check_type	Idea	Input_Problem_Statement
	2	Idea_or_Document	Idea	Update_Ideation
	2	Idea_category	Utility	Categorize_as_Utility
	2	Idea_Category1	System	Categorize_as_System
	2	idea_category3	Anticipative	Categorize_as_Anticipative
	2	Score_or_Save	Save	Save_as_Dreft4
3	3	Check_user_type	Inventor_or_Co-Inventor	My_Idea
	3	check_type	Idea	Input_Problem_Statement
	3	Idea_or_Document	Idea	Update_Ideation
	3	Idea_category	Utility	Categorize_as_Utility
	3	Idea_Category1	System	Categorize_as_System
	3	idea_category3	Anticipative	Categorize_as_Anticipative
	3	Score_or_Save	CIM_score	View_Score
4	4	Check_user_type	Inventor_or_Co-Inventor	My_Idea
	4	check_type	Idea	Input_Problem_Statement
	4	Idea_or_Document	Idea	Update_Ideation
	4	Idea_category	Utility	Categorize_as_Utility
	4	Idea_Category1	System	Categorize_as_System
	4	idea_category3	Agile	Categorize_as_Agile
	4	Score_or_Save	Save	Save_as_Dreft4
5	5	Check_user_type	Inventor_or_Co-Inventor	My_Idea
	5	check_type	Idea	Input_Problem_Statement

Figure 5.5: A Snap Shot of Generated System Test Case

5.3 Comparison with the Industry Test Cases

Many of the testing based on UML do not incorporate test case generation. This has become a matter of consideration now [10]. In the other hand, generating valid test data has become a great achievement. The test requirements were in the form of a possible execution sequence of use cases, messages, transitions and so on, that must be satisfied or covered during testing [17]. here we have generated 1944 system test cases for the application for which system test cases has been designed in the industry. We will represent the deviation of the count of the system test cases designed in the proposed approach and the same designed in the industry. In industry they have designed 3630 system test cases for the same application for which we have optimized the number of system test cases generated by our approach. the percentage of reduction is 46.44%. The details is represented in the following table.

	$Proposed Approach$	$Industry Approach$
No. of System Test cases	1944	3630

Table 5.1: Result

In this we have generated 1944 test cases from 100 test paths which covers all the generated paths that assures 100% branch coverage. Hence our generated test cases are optimal and efficient with respect to the traditional approach followed by the Industry.

Chapter 6

Conclusions

In this thesis, we have basically focused on front end development of the Collaboration Invention Mining module using Adobe Flex 3.0 technology and automatic test paths and test case generation from from the Activity Diagram of the module. It is an automatic approach to generate the test case from the design documents. The framework reported in this thesis is summarized in this chapter. In chapter 5.1, we conclude with the development of the developed software with our contributions to it. In chapter 5.2 we conclude our proposed framework that has been discussed in aforementioned chapter. Finally, we discussed the scope and possible future work of our thesis in chapter 5.3.

6.1 Contribution to Front End Development of CIM

A focused program coupled with management commitment to encourage Inventions (and Innovations) is essential for survival of any business. The outputs need to be protected and monetized through a mature IPR strategy. Collaborative Invention Mining is the process of transforming A Concept or An Idea in an enterprise through a collaborative deliberation into a sustainable invention, applying the EA3 Business Principle. Here we have implemented the

front end of the application using Adobe Flex 3.0 technology. Almost 25 user interfaces have been developed for collaborative invention mining (CIM). Apart from that we were involved in the requirement analysis and design phase with the team in the industry. Finally, we developed almost 3600 system test cases for the application to execute it sot system testing. The testing activities were done with the help of a tool called Application Life Cycle Management (ALM), A tool developed by Tata Consultancy Services for smooth conduct of Testing process.

6.2 Contribution to Test Case Generation for CIM

In chapter 4, we consider test requirements as a part of a test case, but our automatically generated test cases specifies an optimized value of test data for which a particular test sequence will be executed together with the expected output. Automatically generated of test cases from activity diagrams plays a vital role in terms of cost and time as it reduces the numbers of test cases by 46.44% satisfying the same criteria, constraints and coverage as the test cases designed by the testing experts in the industry manually. So it can be considered as an effective procedure to design system test cases.

We have defined a methodology that comprises of two algorithms called as Activity Test Path (ATP) and Test Path Traversal (TPT) to generate the test cases from the design document, i.e., activity diagram of an application. To validate the intermediate representation, we have used a tool called GraphViz Editor, which generate the intermediate graph corresponding to the activity diagram. As it produces a minimized set of test cases, it can be considered as an optimal approach for the system test case generation in industry perspective.

6.3 Scope and Future work

In the future, we will try to optimize the number of system test cases for forthcoming modules. We will also try to prioritize the test cases according to their severity and criticality by using Artificial Intelligence techniques such as Genetic Algorithm Particle Swarm optimization etc.

Bibliography

[1] Anderson, Jim, et al. "Collaborative internet data mining systems." U.S. Patent No. 5,918,010. 29 Jun. 1999.

[2] [2] Mingsong, Chen, Qiu Xiaokang, and Li Xuandong. "Automatic test case generation for UML activity diagrams." Proceedings of the 2006 international workshop on Automation of software test. ACM, 2006.

[3] Chen, Mingsong, et al. "UML activity diagram-based automatic test case generation for Java programs." The Computer Journal 52.5 (2009): 545-556.

[4] Kundu, Debasish, and Debasis Samanta. "A Novel Approach to Generate Test Cases from UML Activity Diagrams." Journal of Object Technology 8.3 (2009): 65-83.

[5] Chen, Mingsong, Prabhat Mishra, and Dhrubajyoti Kalita. "Coverage-driven automatic test generation for UML activity diagrams." Proceedings of the 18th ACM Great Lakes symposium on VLSI. ACM, 2008.

[6] Xu, Dong, Huaizhong Li, and Chiou Peng Lam. "Using adaptive agents to automatically generate test scenarios from the UML activity diagrams."Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific. IEEE, 2005.

[7] Heinecke, Andreas, et al. "Generating test plans for acceptance tests from uml activity diagrams." Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on. IEEE, 2010.

[8] Swain, Santosh Kumar, Durga Prasad Mohapatra, and Rajib Mall. "Test Case Generation Based on State and Activity Models." Journal of Object Technology9.5 (2010): 1-27.

[9] Kim, Hyungchoul, et al. "Test cases generation from UML activity diagrams."Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on. Vol. 3. IEEE, 2007.

[10] Samuel, Philip, R. Mall, and Ajay Kumar Bothra. "Automatic test case generation using unified modeling language (UML) state diagrams." IET software 2.2 (2008): 79-93.

[11] Samuel, Philip, Rajib Mall, and Pratyush Kanth. "Automatic test case generation from UML communication diagrams." Information and software technology 49.2 (2007): 158-171.

[12] Apfelbaum, Larry, and John Doyle. "Model based testing." Software Quality Week Conference. 1997.

[13] Utting, Mark, and Bruno Legeard. Practical model-based testing: a tools approach. Morgan Kaufmann, 2010.

[14] Apfelbaum, Larry, and John Doyle. "Model based testing." Software Quality Week Conference. 1997.

[15] Booch, Grady, James Rumbaugh, and Ivar Jacobson. The unified modeling language user guide. Pearson Education India, 1999.

[16] Aiken, Leona S., and Stephen G. West. Multiple regression: Testing and interpreting interactions. Sage, 1991.

[17] Briand, Lionel, and Yvan Labiche. "A UML-based approach to system testing." Software and Systems Modeling 1.1 (2002): 10-42.